ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Automatic Control Laboratory

Bachelor's Thesis

Modernization of the Ball-on-a-Plate System

Simon Frölich
February 15, 2024

Advisors
Niklas Schmid, Jiaqi Yan

Chapter 1

Abstract

This thesis presents the modernization of a Ball-on-a-Plate system, originally developed in 2010 at ETH Zurich. At its core, the system comprises a dynamically tiltable plate, controlled by motors to maneuver a ball across its surface. Ball-on-a-Plate systems provide an excellent platform for understanding and demonstrating control engineering principles. The project's primary aim is to upgrade the system's outdated hardware and software, enhancing its capabilities and user-friendliness. Key modifications include replacing the original controller with a Raspberry Pi, integrating a new high-resolution camera, and implementing advanced control algorithms. These enhancements enable more accurate ball tracking and the implementation of efficient control strategies, such as PID, LQR, and MPC. A novel aspect of this work is the introduction of a haptic interface using a PS4 controller, offering an intuitive and interactive way to control the system. Through the the realization of a ball position controller the modernized system was put to the test. The results indicate successful achievement of several predefined control objectives, including precise ball position regulation and effective trajectory tracking, confirming the system's enhanced capabilities and potential for further educational and experimental applications.

Chapter 2

Preface

Embarking on this journey to complete my bachelor's thesis on the modernization of the Ball-on-a-Plate System has been an enlightening and transformative experience. I extend my deepest gratitude to my supervisors, Niklas Schmid and Jiaqi Yan, whose guidance, expertise, and patience have been invaluable throughout this process.

I am also profoundly thankful to the institute and the ETH Zurich for providing an environment conducive to learning and exploration, and for the resources and support that have been fundamental to my project.

This journey would not have been possible without the encouragement and support from friends, family, and colleagues. Their belief in me has been a constant source of motivation and strength.

Working on this project has been a significant milestone in my academic and personal development. The challenges encountered have fostered a deeper understanding of control systems, programming, and engineering, enriching my skills and knowledge in ways I had not anticipated. This thesis not only marks the culmination of my undergraduate studies but also lays a foundation for my future endeavors in the field of engineering.

As I close this chapter and look forward to the next, I carry with me not only the knowledge gained but also the memories of this incredible journey.

Contents

List of Figures

List of Tables

Chapter 3

Introduction

3.1 Background and Motivation

This thesis focuses on the Ball-on-a-Plate system built in 2010 in the context of a Master thesis at ETH Zurich [9]. Essentially, the system consists of a white plate which can be independently tilted in two directions by one motor each and a black ball, which is positioned on the plate. The objective is to manipulate the plate's angle to control the ball's position. Two multi-turn potentiometers that turn with the plate provide us with measurements, which can be converted into angles (degrees). A camera is fitted above the plate to record the ball and thus track its position. Via a router communication between the system and another computer can be established, allowing us to run complex control algorithms using programs like MATLAB. The system provides an excellent platform for understanding and demonstrating key principles in control engineering such as feedback loops, PID control and instability. Originally designed for students to conduct lab experiments and test out different control algorithms, it is now outdated and user-unfriendly. Since the control unit (X20 system from B & R Automation) does not have its own interface, trying out new control algorithms is very cumbersome because the controller is changed by flashing C-code to it. In addition, the CPU has very limited processing power compared to modern standards, potentially inhibiting us from running more complex control algorithms.

3.2 Objectives of the Thesis

The goal of this Bachelor's thesis is to completely modernize the system by removing the existing controller and replacing it with a Raspberry Pi. This allows us to directly program on the Raspberry Pi with Python. Additionally, installing a newer camera will result in more accurate tracking of the position and more importantly a higher frame rate. It should also be able to interface between a computer connected to the router. This will enable us to run the control algorithms on the computer and make use of its superior processing power compared to the Raspberry Pi.
Regarding the software of the system, a first objective is to implement a haptic interface with which one can control the plate using a PS4-Controller. This is merely to replace the already existing haptic interface and making easier and more intuitive to control (through a Joystick which can be tilted in two directions instead of turning two knobs). The haptic interface serves as fun way to interact with the system and explore the dynamics of controlling the ball. In the future it could probably be used for playing games, too.
A second objective is to implement a control algorithm on the renewed hardware to regulate the ball to the middle position of the plate. This should be done with a steady state-error of

less than 1cm. A further goal is to let the ball track a circle within ± 4 cm. Different control strategies like PID (Proportional, Integral, Derivative), LQR (Linear Quadratic Regulator) and MPC (Model Predictive Control) can be tested and compared with each other.

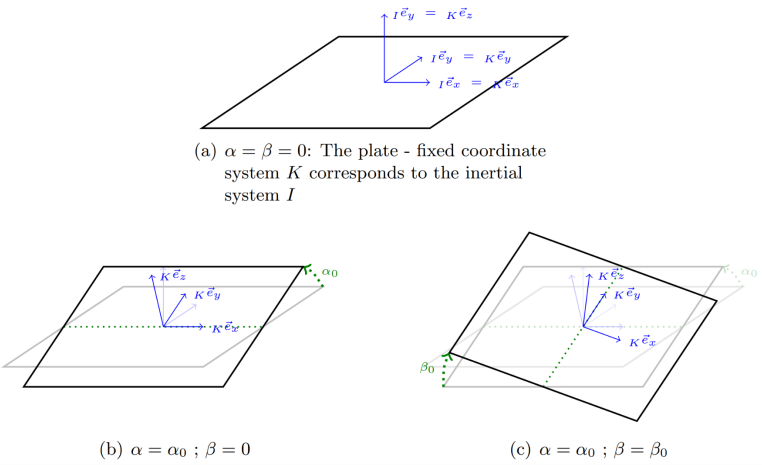
Chapter 4

Theoretical Background

4.1 Ball-on-a-Plate Systems

This subsection discusses the mathematical model of the Ball-on-a-Plate System, described in the original Master thesis [9]. For a complete and comprehensive overview of the system, see Section 5.1.

Figure 4.1 shows the plate-fixed coordinate system with the corresponding angular positions.



(a) $\alpha = \beta = 0$: The plate - fixed coordinate system K corresponds to the inertial system I

(b) $\alpha = \alpha_0$; $\beta = 0$

(c) $\alpha = \alpha_0$; $\beta = \beta_0$

Figure 4.1: Plate coordinate system with angular positions (adapted from [9])

Motor/Plate Model

The Motor/Plate model relates the (effective) applied motor voltage U_M and angular position of the plate. In space-state representation the model for the two angular positions of the plate is described as follows:

$$\begin{bmatrix} \dot{\alpha} \\ \ddot{\alpha} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -b_\alpha \end{bmatrix} \cdot \begin{bmatrix} \alpha \\ \dot{\alpha} \end{bmatrix} + \begin{bmatrix} 0 \\ a_\alpha \end{bmatrix} \cdot U_{M_\alpha} \tag{4.1}$$

for α and:

$$\begin{bmatrix} \dot{\beta} \\ \ddot{\beta} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -b_\beta \end{bmatrix} \cdot \begin{bmatrix} \beta \\ \dot{\beta} \end{bmatrix} + \begin{bmatrix} 0 \\ a_\beta \end{bmatrix} \cdot U_{M_\beta} \tag{4.2}$$

for β as introduced in Fig. 4.1 with U_{M_α} and U_{M_β} representing the applied voltage at the motor for the corresponding angular positions. Since we can measure the angles of the plate with a potentiometer the output equations follow as

$$y_\alpha = \begin{bmatrix} 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \alpha \\ \dot{\alpha} \end{bmatrix}, \qquad y_\beta = \begin{bmatrix} 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \beta \\ \dot{\beta} \end{bmatrix}. \tag{4.3}$$

The coefficients featured in the equations represent these values:

$$a_\alpha = 3.1470 \left[\tfrac{rad}{V \cdot s^2}\right], \qquad a_\beta = 3.1480 \left[\tfrac{rad}{V \cdot s^2}\right] \tag{4.4}$$

$$b_\alpha = 44.6252 \left[\tfrac{1}{s}\right], \qquad b_\beta = 42.4513 \left[\tfrac{1}{s}\right]. \tag{4.5}$$

Ball Model

To relate the position of the ball (x_B, y_B) with the angular positions (α, β) a ball model is introduced. The ball model after linearization can separated into two separate parts [9]. A system with input β and states x, $\dot{x}$ which yields the state-space representation:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{5}{7} \cdot g \end{bmatrix} \cdot \beta \tag{4.6}$$

$$x_B = \begin{bmatrix} 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ \dot{x} \end{bmatrix}, \tag{4.7}$$

and a system with input α and states y, $\dot{y}$:

$$\begin{bmatrix} \dot{y} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} y \\ \dot{y} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{5}{7} \cdot g \end{bmatrix} \cdot \alpha \tag{4.8}$$

$$y_B = \begin{bmatrix} 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} y \\ \dot{y} \end{bmatrix} \tag{4.9}$$

with x and y representing the coordinates introduced in Figure 4.1.

4.2 Control Theory

In this section a quick overview of the control theory used is given.
Control theory is a field of engineering and mathematics that deals with manipulating dynamical systems using feedback. Basically, the objective is to regulate some (or multiple) system variable to a desired state [5].

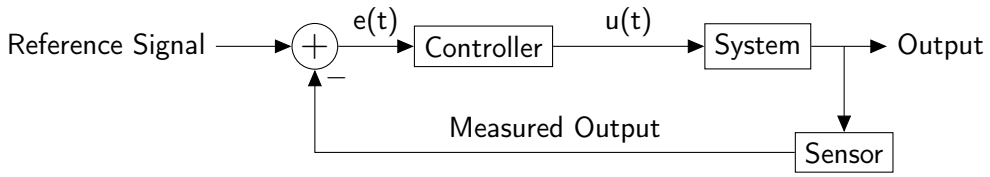


Figure 4.2: A Simple Feedback Control Loop

In Figure 4.2 a simple feedback loop can be seen. By continually subtracting the measured output value of the system from the reference signal, i.e. the desired output value, it obtains the error e(t). Based on that error the Controller then calculates the actuator signal u(t), in order to regulate the error to zero. In our case that means the controller will turn the plate based on the difference between the position of the ball and our desired position (reference signal).

4.2.1 PID Control

PID Control is the most common strategy used in control systems due to its wide range of applications and intuitive nature. In most tasks a PID Controller is sufficient in attaining a generally good performance. PID stands for Proportional-Integral-Derivative, which are the three basic coefficients in this algorithm. In essence, it calculates the output by multiplying the error with the proportional coefficient, the accumulated error with the integral coefficient and the rate of change of the error with the derivative coefficient. Adding all these together gives then the output of the PID. A PID controller can be mathematically represented as:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt} e(t)$$

with K_p representing the proportional, K_i the integral and K_d the derivative coefficient. By tuning these three parameters, one can optimize the controller for desired performance objectives like rise-time, steady-state error, robustness against measurement noise and more.

Chapter 5

Design and Implementation

To give the reader a better overview over the whole system, Figure 5.1 shows the (modernized) system with key components labeled.

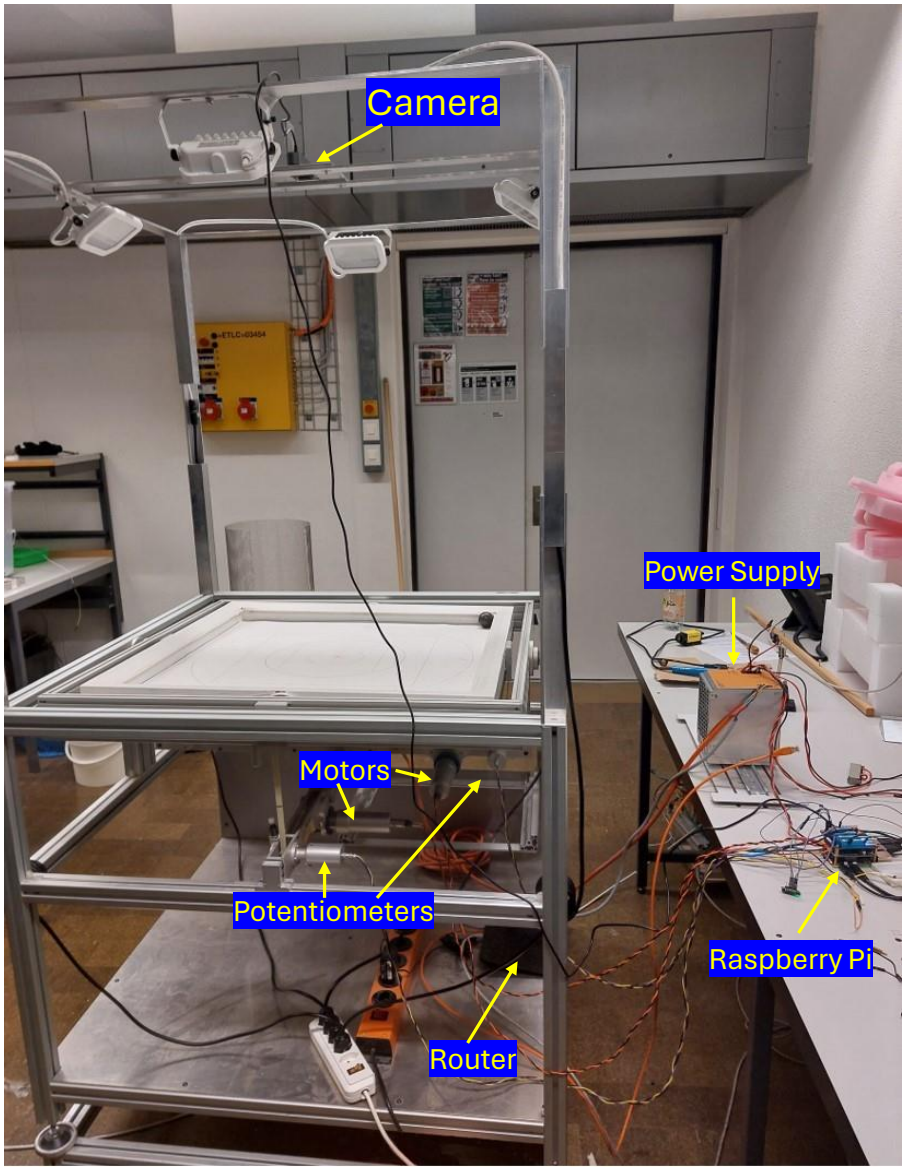


Figure 5.1: Modernized Ball-on-a-Plate System

A detailed description how all the cables are connected can be found under Section 5.2.

5.1 Component Overview

This section will give an overview of all newly installed components, as well as the old components that were decided to be kept in the system. Data sheets for the relevant components are attached to this thesis.

Actuators

Since the motors are still functional, there was no need to replace them. Each of the two plates is actuated by a brushed DC micromotor 3557K024CR by Faulhaber. It takes 24V DC and can be controlled via PWM (Pulse Width Modulation). A planetary gearhead with translation 14:1 reduces the gear ratio between the plate and the motor crank. This is necessary because the speed of the motor would otherwise be too fast for the plate.

Power Supply

There was no reason to install a new power supply into the system. The original power supply, the 0PS1200.1 from B & R [2], is still working without any issue and supplies the desired 24V DC at up to 20A current.

Sensors

To measure the plate's angles two multi-turn 10 kΩ potentiometers (model 534) by Spectrol [8] are installed. Basically, potentiometers are adjustable voltage dividers. When the plate tilts, the potentiometers are turning due to being connected to the V-belt driving the plates from the motors. Therefore, the potentiometer's resistances are dependant on the angle of the plate. After powering the potentiometers, and reading the voltage difference between the ground and the wiper (the voltage divider), we convert these values into angles (see Section 5.3.1).

Raspberry Pi

The center piece of the whole system is a Raspberry Pi 5, a small single-board computer. What makes the Raspberry particularly useful for these kind of applications are its GPIO (General Purpose In Out) Pins. With these we can control the speed of the motor by giving signals to the motor drivers (see Section 5.1). And above all, it makes it into a stand-alone system without the need of an external computer. All the components (Actuators, Sensors, Network Connection) can be connected to and controlled by the Raspberry Pi. Other advantages of the Raspberry Pi compared to the old X20 CPU are the affordability, compact size, superior computing power and its flexibility when using programming languages [3]. There exists a extensive community support for the Raspberry Pi OS and a wide range of software libraries for all kinds of programming languages, which makes the development process significantly easier. For all the above mentioned reasons the Raspberry Pi proved to be an excellent choice for the Ball-on-a-Plate System.

Motor Drivers

It is not possible to directly power the motors with the Raspberry's GPIO pins because they would not be able to provide sufficient current. Therefore, we need to utilize motor drivers. A Motor driver is an electrical component that supplies current from the power supply to the

motor. Since we are using brushed DC motors, we can only regulate the speed of the motor by supplying more or less current. The motor driver achieves this by using Pulse Width Modulation (PWM). PWM controls the motor's speed by varying the amount of time the voltage is applied to the motor. By rapidly switching the voltage on and off, the effective power to the motor is controlled, thus controlling the speed.
A low-power PWM signal from the Raspberry Pi's GPIO pins is used to the control the motor driver's PWM output. Additionally, the motor driver should have an H-bridge to change the polarity of the outputs in order to change the direction of the motor.
Based on all those specifications, the motor driver module L298N was chosen initially. It is very inexpensive and able to output voltages from 5V to 35V at a maximum current of 2A. However, in practice it proved to be unsuitable because there are no safety mechanism against back EMF i.e., when the motor changes its direction too quickly, inducing high voltages which destroy the module. In the end a good solution was the Waveshare Raspberry Pi motor driver board [10]. It is a GPIO extension board meaning we can directly put it on the Raspberry Pi. This means we do not need additional jumper wires. Capable of supplying 5 A per motor, which is more than enough for our needs, this board also includes plenty of safety mechanisms to protect itself and more importantly the Raspberry Pi. Figure 5.2 shows the motor driver board on top of the Raspberry Pi.

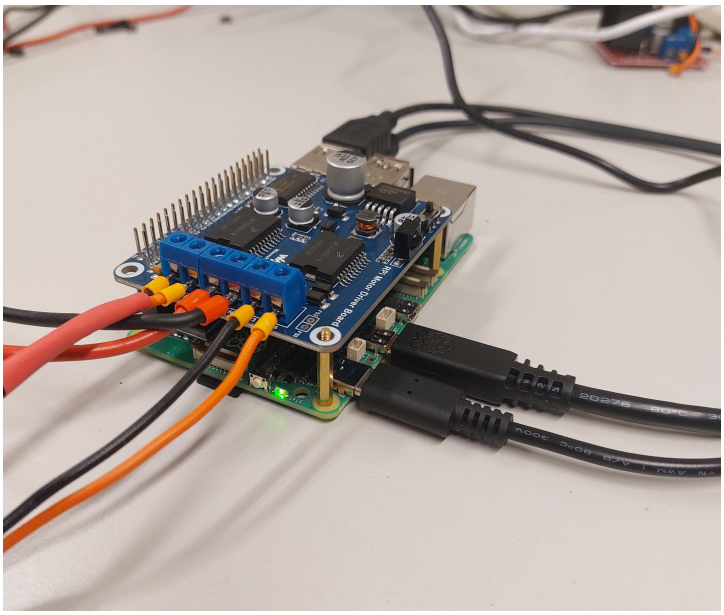


Figure 5.2: Motor Driver Board on Raspberry Pi

Camera

As mentioned previously, the camera (A Cognex In Sight 1050) was quite outdated. Due to only providing a resolution of 640x480 pixel and not being easily connectable to the Raspberry Pi, the decision was easy to implement a new camera. Ultimately it was decided to use the Logitech Brio Ultra HD webcam. Not only does it provide a high resolution and frame-rate, but it is also relatively easy to connect it directly to the Raspberry Pi via a standard USB-C to USB-A cable [7].

Analog-to-Digital Converter

In order to measure the voltage drop over the potentiometers, it is necessary to convert this analog signal into a digital signal which can then be used to calculate the angle of the plate. Unfortunately, the Raspberry Pi does not have a built-in Analog-to-Digital Converter (ADC). Hence, an external ADC is necessary which reads the potentiometer voltage, converts it into a digital signal and sends it to the Raspberry Pi.
We decided to use the ADS1115 ADC, as it provides a sample rate of up to 860 samples/second at a high resolution of 16-Bit [6]. Up to 4 input channels can be connected at once.

Raspberry Pi Screw Terminal Hat

Because we want to power the potentiometers directly from the Raspberry Pi, a convenient solution is the Raspberry Pi Screw Terminal Hat [1]. It can be sticked on the GPIO pins and provides us with screw terminals for every GPIO pin. Basically, screw terminals are a more secure way of connecting wires and this way we are able to connect the potentiometers' wires directly to the Raspberry Pi.

5.2 Electrical Design and Assembly

This subsection discusses how all the components are connected with each other and how everything was assembled.
Using the motor driver board and screw terminal hat described earlier reduces the number of necessary cables. Both modules are stacked on top of each other as shown in Figure 5.3. This means we do not need any jumper wires for the motor driver like if we had used a standard motor driver module like the L298N. We only need to plug the power supply cables and the cables to the motor into the screw terminals on the motor driver board. Subsequently, we can control the motors with signals from the Raspberry Pi's GPIO pins.

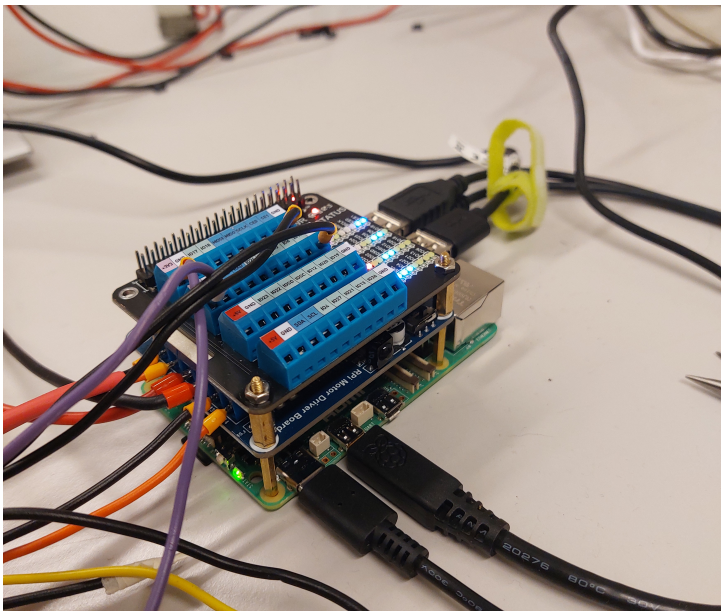


Figure 5.3: Screw Terminal Hat stacked on top of Motor Driver Board

As mentioned earlier, the potentiometers are powered directly with the Raspberry Pi as it is capable of delivering 3.3V DC at low currents via GPIO pins. For that we need to connect both

the potentiometer power cables to the Raspberry Pi ground and 3.3V pin such that current flows through the potentiometer. The cable connected to the wiper of the potentiometer goes to the ADC.
Other connections to the ADCs (we need one for each direction, see Section 5.3.1) include GND, 3.3V power supply as well as SDA and SCL pins for I2C communication between ADC and Raspberry Pi. To effectively utilize two ADCs for each directional measurement in the system, it's essential to establish distinct connections for each ADC. This is achieved by linking either the 3.3V pin or the ground pin to the ADDR (address) pin of each respective ADC. This configuration ensures that each ADC is assigned a unique address, enabling the Raspberry Pi to accurately distinguish and store the values from each ADC in the correct register. Figure 5.4 illustrates how the components are connected with each other.

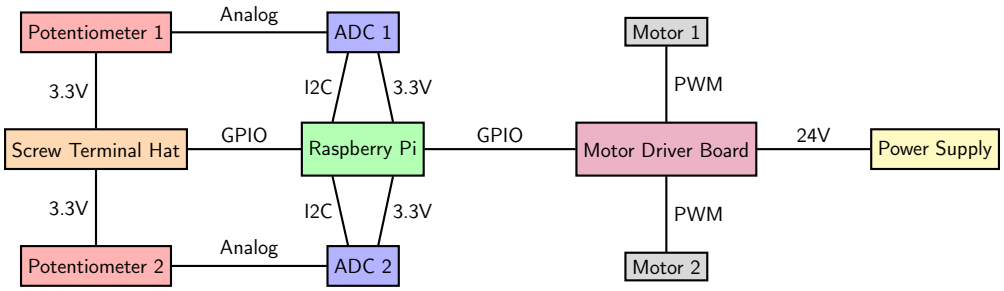


Figure 5.4: Connections between components

Connecting the camera is easy. A USB-A to USB-C cable is sufficient, with the USB-A going into the Raspberry Pi and the USB-C into the camera.
Lastly, in order to establish communication with an external computer, an Ethernet cable from Raspberry Pi to the router is needed. The router produces its own internet that, when both the Raspberry Pi and the other PC are connected, serves as communication channel. This will be done using sockets (Section 5.3.5). Even though the Ethernet cables are technically not necessary, since both computer and RPi can also connect to the internet via WiFi, Ethernet cables obviously have significantly lower network latency.

5.3 Software Development

This subsection discusses the software used for all tasks of the project.

Python is pre-installed on the Raspberry Pi OS, and has an unmatched list of modules and extensive community support. Hence it is the programming language of choice.

5.3.1 Sensors

Camera Implementation and Image Processing

The Ball-on-a-Plate system utilizes the Logitech Brio Ultra HD webcam, interfaced with the Raspberry Pi 5, for real-time ball tracking. This segment describes the implementation of the camera module and the image processing approach, as coded in Python using OpenCV. OpenCV is a particularly useful library as it allows us to both control external cameras and process the images.

Code Structure A rough outline of the structure of the script used to detect the ball's position and speed is given by the pseudo code below (see Algorithm 1). The complete code can be found in the documents attached to the thesis.

Algorithm 1 Ball Position and Speed Detection

procedure DETECTPOSITION(frame)
    Crop frame
    Convert frame to gray scale
    Apply Threshold
    Filter Noise
    Detect ball and calculate position
    Return ball position
end procedure
procedure CAPTUREVIDEO
    Initialize camera and set parameters
    while True **do**
        $currentTime \leftarrow Time$
        $frame \leftarrow ReadFrame$
        $currentPos \leftarrow DetectPosition(frame)$
        $Speed \leftarrow \frac{currentPos - previousPos}{currentTime - previousTime}$
        $previousPos \leftarrow currentPos$
        $previousTime \leftarrow currentTime$
        $currentPos \leftarrow AlignPositionWithCoordinateFrame(currentPos)$
        $currentPos \leftarrow ConvertToMeters(currentPos)$
    end while
end procedure

Camera Setup and Configuration Before capturing images it is necessary to specify the intended configuration of the camera to optimize performance. We can set the frame rate per second, frame width and frame height, as well as video capture format. However, only combinations that are supported by the camera's corresponding Linux driver (V4L2) are possible. In the end a frame rate of 60 FPS and a 1280x720 pixel resolution with the video capture format Motion

JPG (MJPG) was chosen. This configuration provides a good balance between image quality and frame rate. Another important setting we need to adjust is the white balance temperature of the camera. Because of the strong lighting the image is too white, which means the camera tries to compensate by making the image more yellowish. This in turn messes with the image processing, so we need to lower the white balance temperature of the camera manually with a Linux command. Subsequently, the image stays bright, which is advantageous for detecting the ball's position.

Image Processing Technique The core of the image processing involves detecting the ball's position in each frame. The steps are as follows:

Frame Cropping: Each frame is cropped to focus on the relevant area of the plate.
Grayscale Conversion and Thresholding: Frames are converted to grayscale and a binary threshold is applied to highlight the black ball against the white plate.
Morphological Operations: To enhance the binary image, morphological operations like opening are used to remove noise.
Contour Detection and Analysis: Contours are detected in the processed image. The ball is identified based on its area, perimeter, and circularity, distinguishing it from potential other objects.
Center Calculation: Once the ball is identified, its center coordinates are calculated using image moments.

Position and Speed Calculation A translation and rotation is applied to the coordinates to align the coordinates with the coordinate vectors drawn on the plate. The position of the ball is translated from pixel coordinates to physical space (in meters). This translation is dependant on the resolution we configured earlier. The script also calculates the ball's speed between consecutive frames, providing essential data for the control algorithms.

Real-time Data Handling The system operates in real-time, capturing and processing frames continuously. Data containing the ball's position and speed are sent to a data queue for further processing by the control algorithm.

Potentiometer Software Implementation

Here is a quick overview of the structure of the code interfacing with the ADC (see Algorithm 2). Complete code can be in the attached files.

Initialization The interfacing of potentiometers with the Raspberry Pi is implemented using the Python library "Adafruit CircuitPython ADS1x15" by controlling the ADS1115 Analog-to-Digital Converter (ADC). The script begins by initializing the I2C bus, which facilitates communication between the Raspberry Pi and the ADC. For continuous data acquisition, the script sets the ADCs to continuous mode with the maximum data rate of 860 samples/second. In this mode the ADC will continuously convert the input and put it into a register. Consequently, the Raspberry Pi does not have to wait for the ADC to finish reading the input and instead always extracts the latest value. One drawback of this mode, however, is that one ADC is limited to one connection i.e., two ADCs are necessary for the two potentiometers. Hence, the register must be specified because there is only one SDA and SCL pin (I2C) on the Raspberry Pi. The register must be specified according to how we connected the ADDR pin of the ADC.

Algorithm 2 Read ADC

```
procedure VoltageToDegrees(voltage)
    Convert voltage to angular degrees
    return degrees
end procedure
procedure InititializeADC
    Initialize I2C
    Create ADS object
    Set ADC mode to continuous
    Set data rate
    return channel for voltage reading
end procedure
procedure ReadADC(channel)
    voltage ← channel.voltage
    return VoltageToDegrees(voltage) // Convert Voltage to Angle
end procedure
```

If the ADDR pin is connected to the 3.3V pin of the Raspberry Pi, the register of that will be 0x49 and 0x48 if it is connected to the Raspberry Pi ground.

Reading Potentiometer After having initialized the ADC, we can simply get the latest value the ADC has read by calling the ReadADC() Function described in the Algorithm 2. This is quite advantageous as we are able to continuously obtain the plate's angles with very little delay.

Calibration The obtained value must be converted from a voltage into its corresponding angle in degrees. This can be done by applying a linear mapping. By measuring the plate's angles with an external device (like a smartphone) and obtaining the ADC's value we can calculate the parameters of this mapping. Only two data points are necessary since we need only a slope and an intercept for our mapping.

5.3.2 Actuators

The motors can be controlled by the Raspberry Pi's GPIO pins via the Waveshare motor driver board as described in Section 5.1. There are a few different Python libraries for Linux that enable us controlling the GPIO pins from code. A good choice is the "gpiozero" library with the "lgpio" module as backend [4]. At time of writing (January 2024), some other popular GPIO libaries (e.g. RPi.GPIO) are not yet compatible with the newly released Rasperry Pi 5 that we are using.

General Principle Each motor has two pins responsible for setting the direction the motor will spin. These will either take a True or False value, i.e. logic high (3.3V) or logic low (0V). Via a third pin that outputs a PWM signal, the speed of the motor is regulated.

Table 5.1: GPIO Pin Assignment for Motor Control

Function	**GPIO Pin (x-Axis)**	**GPIO Pin (y-Axis)**
Direction Pin 1	20	6
Direction Pin 2	21	13
PWM Control	26	12

Table 5.1 shows which GPIO pins are used. X-and Y-axis refer to the axis drawn on the plate, i.e. the axis which is rotated around the other axis. If we set Direction Pin 1 to True and Direction Pin 2 to False, the motor will spin forward and the other way around for reverse. "Forward" in this case is defined as the direction in which the plate's angle will increase in degrees.
An intuitive way to implement this in Python is by utilizing classes. They facilitate initialization, changing pin values, and releasing GPIO resources.

Initialization of Motors The MotorControl class is initialized with a specific axis (1 or 2), corresponding to the motor it controls, where 1 stands for the x-axis and 2 for the y-axis as illustrated in Table 5.1. With the "OutputDevice" and "PWMOutputDevice" classes from "gpiozero" pin objects are created. For the PWM pin a frequency has to be specified. A good value proved to be 500Hz. Additionally, a so called pin factory (i.e. the backend) has to be given as an argument since the (default) backend library RPi.GPIO is yet not compatible with Raspberry Pi 5.

Motor Control Logic The class provides an update method that takes a control input and the current angle of the plate. The polarity of the control input determines the motor's direction, with positive equating to forward. Since the control inputs represents a duty cycle, only values ranging from -100 to 100 are acceptable. This will be limited already in the control loop. A safety feature is incorporated to prevent the motor from moving if the plate's angle exceeds a predefined threshold.

Speed Control The update_motor method sets the direction of the motor by activating the appropriate GPIO pins and controls the speed using PWM. Calling the pin.on() method sets the voltage to high and the pin.off() to low. The speed value, a fraction between 0 and 1, represents the duty cycle of the PWM signal, thereby controlling the motor's speed. Therefore, it is necessary to first divide the absolute value of the control input by 100. The PWM pin's value is then changed to the appropriate speed.

Resource Management The class includes a clean_up method to properly release the GPIO resources when the motor control is no longer needed. This is needed to turn off all pins properly when the program is stopped.

5.3.3 Manual Control

Now that we can interface with the motors and measure the plate's angles, we can implement the haptic interface. Surprisingly, this can be done in a relatively simple and straight-forward way with a PS4-Controller. The PS4-Controller can be wirelessly connected to the Raspberry Pi via Bluetooth without the need to download additional drivers. Through the Python library "pygame" the controller's joystick position can be obtained. The idea is to tilt the plate in the same direction the joystick is moved, providing an intuitive way to control the system.

Code Implementation If a controller is connected to the Raspberry Pi, the "pygame" library lets us create a joystick object. We then select only axis 3 and 4 representing the axis of the right joystick. We use the "MotorControl" class discussed in the previous section to initialize the motors' interface.
In a while True loop the *pygame.pump* function is repeatedly called to update the values of the joystick. These values, ranging from -1 to 1, can then be extracted. We may need to multiply the axis with -1 or swap the axis, depending on where we are standing, such that controlling it remains intuitive. E.g., when we move the joystick to the right, the plate from our standpoint

should tilt to the right too. Since the joystick values lie between -1 and 1, it is necessary to multiply them by some factor (typically less than 10) to allow quicker movement of the plate. With each iteration of the loop the speed and direction of the motors is updated by calling the MotorControl.update method.

It has to be considered, however, that by moving the joystick the plate's angular velocities are controlled and not the angular positions. This is in contrast to the previous haptic interface where you could directly control the plate's angular positions. Though this could be implemented in software, it is not in the scope of this thesis. Trying to get the ball to the center of the plate is a fun mini game to better understand the dynamics of the system.

5.3.4 Automatic Control

As mentioned in the introduction, one goal of this Bachelor's thesis is to implement a control algorithm to regulate the ball's position on the plate. For this purpose we will need two loops, an outer and an inner control loop. The outer control loop will be responsible for giving a reference angle to the inner loop. Since these are not stepper motors, the inner loop is necessary to control the angle of the plate by applying the appropriate motor speed. Figure 5.5 illustrates the concept of this complete control loop.

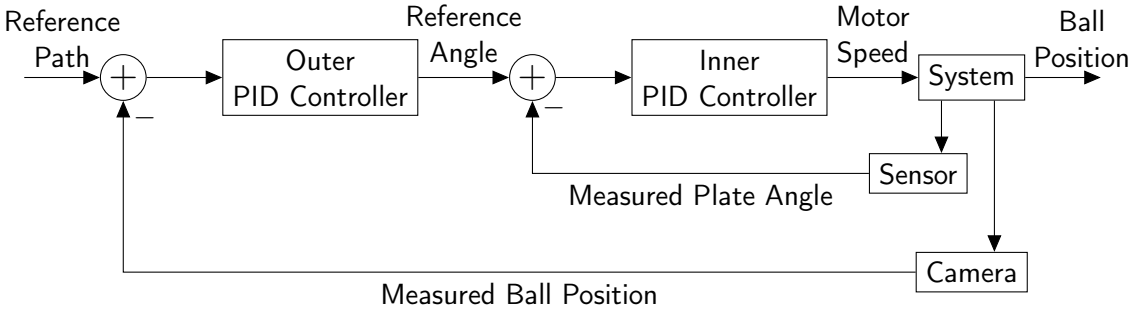


Figure 5.5: Ball Position Control Loop

In Section 4.1 a linearized model of the plate dynamics was introduced which showed that both axis of rotation do not interact with each other. Though in the real and non-linear system this is generally not the case. However, we can assume this interaction to be negligible for small angles. Hence, two separate ball control loops are used, with one responsible for the x-coordinate and one for the y-coordinate.

Software Implementation

In the code this control loop, however, will be divided into three different loops. It utilizes multi-threading to manage the three primary functions simultaneously: video capture, ball position control and angle control. Because these loops run on different frequencies, multi-threading is crucial to ensure real-time control without substantial delays. As explained above, there will be one controller for each coordinate (x and y) in the angle and ball position loops, whose response will be calculated successively.

Video Capture Loop As described in Section 5.3.1, this thread is dedicated to video capture, where the position of the ball is continuously recorded, its position and speed calculated, and fed into a data queue. The data queue acts as intermediary storage, allowing the data to be accessed by the second thread - the ball position loop.

Ball Position Loop This loop uses two PID controllers to calculate the reference angles needed to achieve the ball's desired position. Hereby, it compares the ball's position with the reference position that we define, and based on that error calculates the response of the PID controllers. Additionally, the ball's speed is inversely taken into account through linear substraction. The reference angles are then communicated to the angle control loop via global variables. A simplified version of this loop is shown in Listing 5.1.

```
from simple_pid import PID
import queue

class LowPassFilter:
    # a simple low-pass filter taking into account
    # previous values

data_queue = queue.Queue()

def ball_position_loop():
    global ref_angle
    # PID control for ball position
    x_ball_pid = PID(Kp, Kd, Ki, setpoint=0)
    #limit the maximum reference angle
    x_ball_pid.output_limits = (-10, 10)
    #Initialize Low-Pass Filter
    lpf = LowPassFilter(0.5)
    while True:
        data = data_queue.get() #retrieve position and speed
        x_pos = data["position"][0] #position x-coordinate
        x_vel = data["speed"][0] #speed x-coordinate
        #apply low-pass filter to smoothen signal:
        filtered_x_vel = lpf.update(x_vel)
        #calculate reference angle for angle control loop:
        ref_angle = x_ball_pid(x_pos) - 12*filtered_x_vel
        sleep(0.01) # sleep for frequency control
```

Listing 5.1: Simplified Ball Position Control Loop

Even though only the x-coordinate is shown in the listing, the controller for the y-coordinate will be almost identical. For the PID controllers the Python library "simple_pid" was used which allows us to create the controller with the coefficients Kp, Kd and Ki. It is also possible to change the setpoint (the desired position) dynamically (e.g. going in a circular patter around the center of the plate). Additionally, a simple low-pass filter is applied to the speed signal to smoothen it and avoid spikes in the motors' movement.

Angle Control Loop Finally, the angle control loop, which also employs PID controller, takes theses reference angles and continually calculates the motor speeds needed to achieve these angles. This loop runs on a high frequency to ensure a smooth and accurate behaviour of the system. A simplified version of this loop is depicted in Listing 5.2.

```
from simple_pid import PID
from potentiometer import read_potentiometer, init_ADC
from "motordriverboard.py" import MotorControl

def angle_control_loop():
    global ref_angle #reference angle from ball position loop
    #Inner PID controller:
    x_pid = PID(7, 0.6, 0.66, setpoint=0)
    x_pid.output_limits = (-10, 10) #Speed limits for safety
    x_motor = MotorControl(1)
```

```
    #Establish connection to ADC and run in continuous mode:
    x_adc = init_ADC(1)
    while True:
        #get plate angle:
        x_angle = read_potentiometer(x_adc, 1)
        #update target angle:
        x_pid.setpoint = ref_angle
        #calculate control signal:
        control_input = x_pid(x_angle)
        #change motor speed:
        x_motor.update(x_control, x_angle)
```

Listing 5.2: Simplified Angle Control Loop

Again, the y-coordinate will be controlled in the same loop but for simplicity's sake it is not included in this exemplary code. The "MotorControl" class, detailed in section 5.3.2, is used to interface with the motors and applying the speed the PID controller calculated. Initialization and reading of the ADC (to get the plate's angles) is done via the functions read_potentiometer and init_ADC, described in Section 5.3.1.

Multi-Threading

In order to run these three loops simultaneously the Python module "threading" was utilized. The program can be shut down by pressing CTRL + C. This code will not be described in this section but can be found along with all other relevant code in the attached files.

5.3.5 Network Communication

A further objective of the thesis was, as laid out in the introduction, to be able to control the system from an external computer. Communication between the Raspberry Pi and an external computer is achieved using socket programming, a method for exchanging data over a network. The Raspberry Pi acts as a server, setting up a socket on a specific IP address and port, and listens for incoming connections from the client (external computer).
The communication process involves multiple threads running concurrently on the Raspberry Pi. This is particularly important to ensure the computer receives the data in real-time, which is necessary for effective and robust control.

Establishing a Connection

Python libraries "socket" and "threading" provide us with ample tools to accomplish this task and are well-documented.
Firstly, both Raspberry Pi and Client computer need to be connected to the same network. In this case the network will be established by the router which was built in context of the Master thesis in 2010 [9]. As choice of network protocol the decision was made to use TCP because it guarantees the arrival of the data on the client side.
Then, the server starts by binding a socket on the IP address the network provides and a port using the socket.bind() method. By calling the socket.listen() method the server listens for attempts made by a client to connect to this socket. Finally, with the socket.accept() method returns the successfully established socket connection. This connection will given to the send and receive threads as function arguments.

Sending and Receiving Data

The first thread is dedicated to capturing video and tracking the ball's position and speed, where the data is continuously pushed into a data queue.

The second thread, known as the send thread, retrieves this data from the queue. Each piece of data, which includes the ball's position and other relevant information, is serialized into JSON format and sent over the network to the connected client. This transmission allows the external computer to receive real-time updates on the ball's position.

```
def send_data(socket, data):
  data = struct.pack('>I', len(data)) + data
  socket.sendall(data)

def send_thread(s):
  while not shutdown:
    time.sleep(0.01)
    if not data_queue.empty():
        data = data_queue.get()
        json_data = json.dumps(data).encode('utf-8')
        send_data(s, json_data)
```

Listing 5.3: Send Thread

Listing 5.3 shows how the send thread works. The function for actually sending the bytes to the client (send_data) packs a struct containing the length of each message in front of every message. That will make sure the client is able to receive the complete message before receiving a new one.
Concurrently, the Raspberry Pi server also runs a receive thread. This thread listens for incoming data from the client, which can be a reference angle or information that the Raspberry Pi might need. By first unpacking the first 4 bytes (the struct) of any data arriving, the length of the income message can be found out, which can be seen in Listing 5.4. Now the thread will listen until the whole message has been received. After receiving the data, it is deserialized from JSON format and processed accordingly.

```
def receive_data(socket):
  raw_len = socket.recv(4)
  if not raw_len:
    return None
  msg_len = struct.unpack('>I', raw_len)[0]
  return socket.recv(msg_len)

def receive_thread(s):
  s.settimeout(1)
  while not shutdown_flag.is_set():
    try:
      recv_data = receive_data(s)
      if recv_data:
        recv_data = json.loads(recv_data.decode('utf-8'))
        print(recv_data)
    except socket.timeout:
      continue
    except OSError:
      break
```

Listing 5.4: Receive Thread

Lastly, to control the system remotely, a fourth thread will be responsible for taking the received control inputs (reference angles) and actually moving the plate. This thread will be identical to the angle control loop shown in Section 5.3.4.

Chapter 6

Testing and Results

This section focuses on the experiments conducted to test the performance of the system.

6.1 Experiments

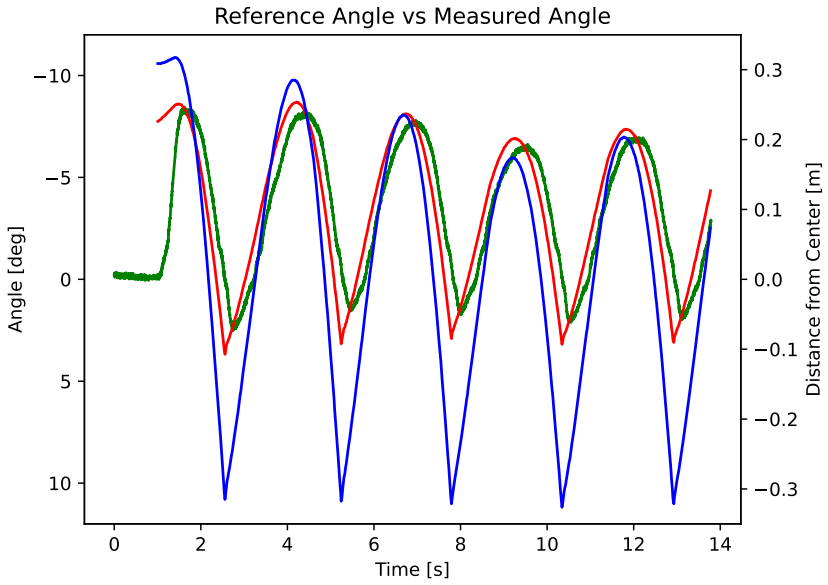
As laid out in the introduction, one objective of this thesis is to regulate the ball's position to the center of the plate. A second objective is to track a circle around the center of plate within $\pm$ 4 cm. In order to achieve this, the parameters of the controllers need to be optimized for our performance goals.

Inner Loop Parameters

First of all, we must find out optimal coefficients for the angle control loop or inner loop. Whether we are using LQR, MPC or PID control for ball position regulation, this loop will always be running in the background and control the plate's angles by driving the motors. Its primary objective is to ensure good tracking of the reference angles i.e., actual angles of the plate should be as close as possible to the angles given by the ball position controller. Since interactions between both angular direction can be neglected (see Section 5.3.4, the PID controllers responsible for turning around x-and y-axis can be designed independently. One experiment to test the performance of the controller parameters was to tilt the plate to the right or left and setting the reference angle to zero. The performance would be evaluated by how fast the plate would achieve this angle (step response). It is also important that there would be no over-shoot to avoid inducing instability. Plotting the trajectory allowed us to check these objectives. By using intuition different PID coefficients were tested and those that yielded the best results were chosen. For both axis the best parameters were $K_p = 7$, $K_i = 0.6$ and $K_d = 0.66$.

Regulating the Ball to Center

Now that we have the parameters for the inner loop, we can focus on ball position control algorithm described in Section 5.3.4. Again, first a PID controller was tried out. Here the goal for the controller was to regulate the ball to the center. Initially the ball would be at rest at an arbitrary position. Afterwards, the control algorithm was started and the ball should be in the center position within at most 15 seconds. At first it was tested without taking the velocity of the ball into account at first. However, it proved to be impossible to control the ball based solely on its position because the momentum of the ball became too high and could not be compensated fast enough. As a result the system became unstable with the ball rolling from one side of the plate to the other. This behavior can be observed in Figure 6.1. Only the x-axis and its corresponding angle are shown.



Measured angle (Green), reference angle (Red), and distance to center (Blue)

Figure 6.1: Behaviour of system without compensating velocity

It became clear that the velocity must be taken into account as well. By simply subtracting the velocity, multiplied by factor, from the output of the PID controller our objective could already be achieved. Countering the velocity of the ball will prevent the ball from gaining too much linear momentum and rolling over the center to the other edge. Through plots of one of the coordinates and just visual observation the performance of the system was judged. Figure 6.2 shows the response of the best performing controller. As can be seen, the experiment resulted in ultimately less than 1cm steady-state error which satisfies our first goal. Because of static friction, a steady-state error of 0cm is almost impossible to attain. In A.1 the parameters that resulted in optimal performance can be found.

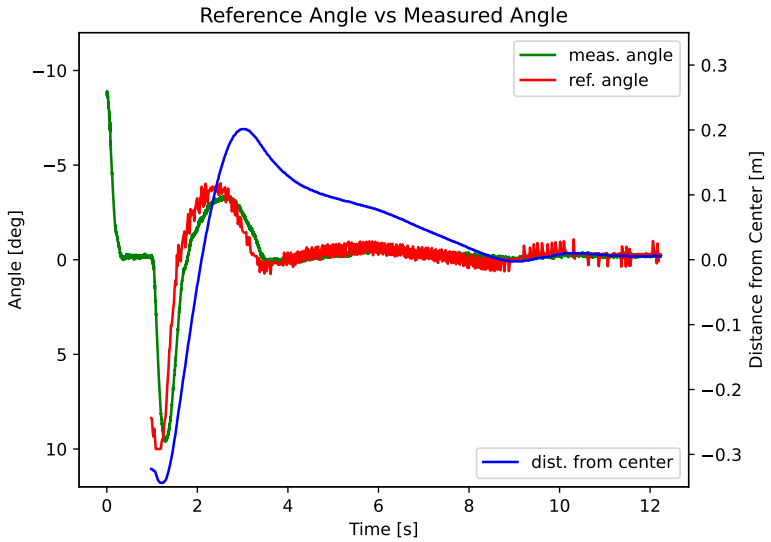


Figure 6.2: Regulating ball to center with velocity compensation.

Circle Tracking

Regarding our second objecting, tracking a circle within ± 4cm, the same controller which was employed to achieve our first objective was utilized. The experiments were started with the ball being at rest at some random location on the plate. More than five circulations were performed to see if the ball would stay inside the boundaries of the circle. Remarkably, the controller proved to be quite capable for this task as well. Though not as precise when using the same parameters, if the parameters are optimized for this task, our second objective can be fulfilled. As can be observed in Figure 6.3 the ball does not deviate from the reference circle more than 4cm. The circle has a radius of 15cm and one circulation takes 2π seconds.

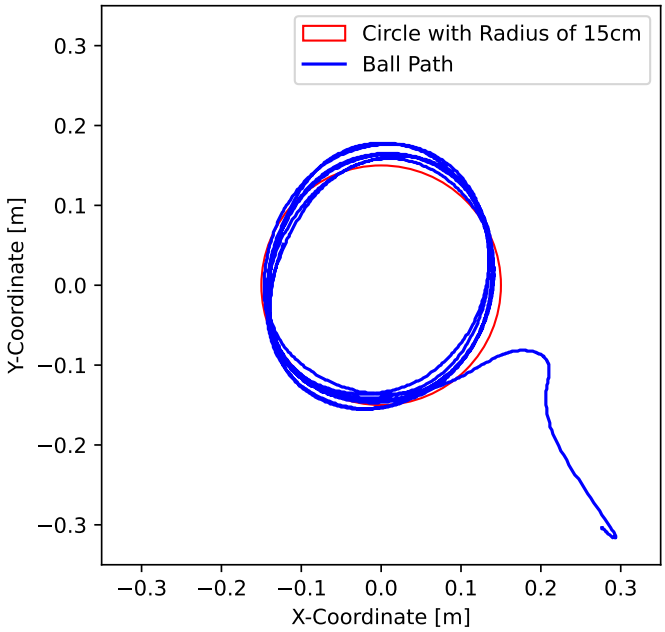


Figure 6.3: Ball Tracking Circle

Figure 6.3 also shows that this works for any starting position of the ball. While satisfying our predifined specifications, Linear Quadratic Regulator (LQR) or Model Predictive Control (MPC) could yield better results for this task because they also account for future reference inputs.

Chapter 7

Conclusion and Future Work

7.1 Summary of Findings

The project's objectives, centered around modernizing and enhancing the functionality of the Ball-on-a-Plate system, were largely met. The system now demonstrates improved performance, greater user accessibility, and enhanced educational value, fulfilling its original purpose as a platform for demonstrating control engineering principles.

Comparing the modernized system to the original system, it is obvious to see the advantages of the new system. The camera has been upgraded with both better resolution and higher frame rate per second. Additionally, the new heart of the system, the Raspberry Pi, is substantially more capable in terms of processing power compared to the old control unit. Another huge improvement is the flexibility of the system. Since the Raspberry Pi provides us with an interface we can directly program on, in any programming language we desire, the system becomes a lot more user-friendly and accessible. This allows us to implement and test new control strategies more easily, and potentially opens the door for other usages such as reinforcement learning. Notably, the new haptic interface using a PS4 controller introduced a more intuitive and interactive way for users to engage with the system.

As for the control objectives, which are outlined in the introduction, they have been achieved as well. By employing PID control and optimizing its coefficients, the control performances have satisfied the predefined specifications.

7.2 Limitations and Challenges

In this section the challenges encountered during this thesis will be discussed as well as the limitations the modernized system has.

Throughout the making of this thesis I have encountered many difficulties and challenges, which I was largely able to solve. These problems let me gain a lot of experience in applied control systems, Linux OS, programming and electrical design.
One limitation of the current system that could be improved upon is the ball detection on near the edges of the plate. Since the camera is only recording a projection of the ball but the plate may be tilted, inaccuracies on the position measurement arise. This could be mathematically solved based on the angles of the plate. Instead of using potentiometers a different kind of sensor may result in a more accurate measurement of the plate's angles.
For improved tracking of reference angles in the angle control loop, another controller like LQR

could be implemented. As mentioned before, for ball position control an MPC or LQR controller could potentially yield better results.

7.3 Outlook

This new system provides a fantastic framework to test out other control strategies and to compare them with the existing ones. Because of the improved user-friendliness, any interested student can now optimize the performance of the system.
As was intended for the original system, this Ball-on-a-Plate system could be made into a laboratory experiment for students to become familiar with different control strategies and provide them with valuable experience in real-world control systems. There are endless possibilities and use cases for this system which can be explored by other users in the future.

Bibliography

[1] 52Pi. *RPi Screw Terminal Hat*. https://wiki.52pi.com/index.php?title=EP-0129. Accessed: 2024-02-14.

[2] B & R Automation. *Power Supply 0PS1200.1*. https://www.br-automation.com/en/products/accessories/power-supplies/single-phase-power-supplies/0ps12001/. Accessed: 2024-02-14.

[3] Raspberry Pi Foundation. *RPi 5 Documentation*. https://www.raspberrypi.com/documentation/computers/raspberry-pi-5.html. Accessed: 2024-02-14.

[4] *gpiozero documentation*. https://gpiozero.readthedocs.io/en/latest/. Accessed: 2024-02-14.

[5] Lino Guzzella. *Analysis and Synthesis of Single-Input Single-Output Control Systems*. vdf Hochschulverlag AG, 2019.

[6] Texas Instruments. *ADS1115 Datasheet*. https://cdn-shop.adafruit.com/datasheets/ads1115.pdf. Accessed: 2024-02-14.

[7] Logitech. *Logitech BRIO Ultra HD*. https://www.logitech.com/en-ch/products/webcams/brio-4k-hdr-webcam.960-001106.html. Accessed: 2024-02-14.

[8] Vishay. *Potentiometer Datasheet*. https://www.vishay.com/docs/57065/533534.pdf. Accessed: 2024-02-14.

[9] René Waldvogel. *MPC Control of a Ball on Plate System*. Master thesis IfA, ETH Zurich. 2010.

[10] Waveshare. *RPi Motor Driver Board*. https://www.waveshare.com/rpi-motor-driver-board.htm. Accessed: 2024-02-14.

Appendix A

Appendix

A.1 Additional Data and Diagrams

Table A.1: Optimal Parameters for Outer Loop for Control Objectives

Objective	K_p	K_i	K_d	**Velocity Factor**
Regulate to Center	20	6	0.1	15
Circle Tracking	20	1	1	20